Porting Indicators Guide

An Application Guide

Table of Contents

Introduction

Software engineers and project managers use a variety of indicators in various phases of software development to track project progress. Porting an application is unique in that most of the code is already written and most of it is probably correct, depending on the quality of the code. Unfortunately, the code that needs to be fixed is generally sprinkled throughout the source code base. So the difficulty in tracking progress is selecting the “right” indicators that are objective, easy to collect, and directly applicable to the porting effort. The purpose of this document is to suggest additional porting indicators that meet those objectives. In addition, a Perl script to help collect data is provided. You are free to use and/or modify the script to suit your specific needs.

Porting Indicators

The following indicators can help monitor your porting progress. The indicators can be generated automatically using the Perl script `Indicators.pl` provided in this application note.

- **Total number of files compiled versus time.** As you begin porting your application, it is unlikely that all of the files will get to the compilation stage on the first try due to earlier build failures. This is especially true for complex applications with custom build steps building many components of the project. This indicator is used to gauge how much of the project has been tackled and to provide a frame of reference with respect to the other indicators. For example, monitoring the number of compilation errors without knowing how much of the application is being compiled is not very useful.
- **Total number of compilation errors versus time.** This indicator tracks the number of compilation errors over time. Eliminating the compilation errors is the first step to getting your application built before you can run it on the Intel® Itanium™ processor or on other IA-64 processors. You should expect that the number of compilation errors will actually increase initially over time. As you resolve the early compilation errors, more and more files will reach the compilation stage. They, in turn, are likely to generate more compilation errors. Once you have peaked on the number of files that must be compiled, then the number of errors will begin to decrease as you resolve each one. You can use this peak as a milestone.
- **Total number of files with compilation errors/warnings versus time.** This indicator provides you with a picture of the distribution of errors and warnings. A low number of files compared to the total number of files compiled is an indication that the errors/warnings are concentrated in just a few areas. Otherwise, a large number of files with errors/warnings as compared to the total number of files in your project indicates that the issues are spread out all over the code.
- **Total number of unique compilation errors/warnings versus time.** Fortunately, the number of errors and warnings is generally not unique. Warnings or errors in header files will be replicated in each file that is compiled with the header files. Therefore, this indicator can provide you with a better picture of the amount of effort needed to resolve all compilation errors and warnings. The assumption is that once you fix the error/warning in the header file, the error/warning will no longer show up in a large number of compilation files. Assuming an equal effort to fix all errors/warnings, this indicator provides a more linear view of the number of warnings/errors to be fixed.

- **Total number of modules linking versus time.** Once the application software compiles, the next step is to monitor the link step. In large applications with many components, it is not unusual to initially have link errors. There are a number of causes for link problems but the most likely issues are missing libraries, lack of IA-64 compatible libraries, or compiler bugs. Knowing the total number of modules that need to be linked, this indicator can be used to determine how close you are to running and debugging on the Itanium processor.

Indicator Perl Script – Indicators.pl

The indicator script, as written, specifically targets applications built using Microsoft Visual Studio* compiling C or C++ code. Microsoft's Visual Studio generates a `.plg` file each time you build a project or compile a file (Under "Options…", "Build" tab, see "Write Build Log" option). It contains a copy of the build results that are displayed in the status window. The `.plg` file contains each error and warning that occurred as well as the file name and line number that it occurred on. The `Indicator.pl` script searches for all `.plg` files in all subdirectories starting at the directory provided as the argument to the script. The script then scans each `.plg` file recording the file name, line number, and error or warning message found. The file names scanned for must have the file extension `.c, .cpp, .cxx, .h, .hpp,` or `.hxx.`

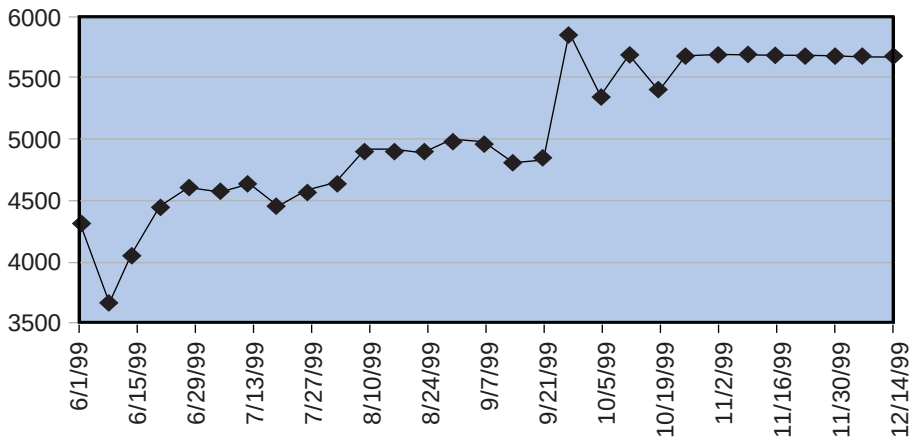
The following table shows the data generated by the indicators.pl script:

Result	Notes
Total Projects Compiled:	Count of the number of `.plg` files processed
Total Compilation Failures due to /FD:	Since the IA-64 compilers do not currently support the Microsoft /FD option for improved file dependency checking, the script detects if this option was set. Unfortunately, this option is not ignored but in fact, the project will fail without compiling any files
Total Files Compiled:	Count of all files compiled with or without errors/warnings
Total File Count with Errors or Warnings:	Count of the number of files with either error or warning messages.
Total Number of Lines with Errors or Warnings:	Count of the number of lines with either error or warning messages.
Total Numbers of Unique Errors or Warnings (multiple E/W on a line):	Count of the number of unique lines with either error or warning messages.
Total Numbers of Unique Warnings (multiple E/W on a line):	Count of the number of unique lines with warning messages.
Total Numbers of Unique Errors (multiple E/W on a line):	Count of the number of unique lines with error messages.

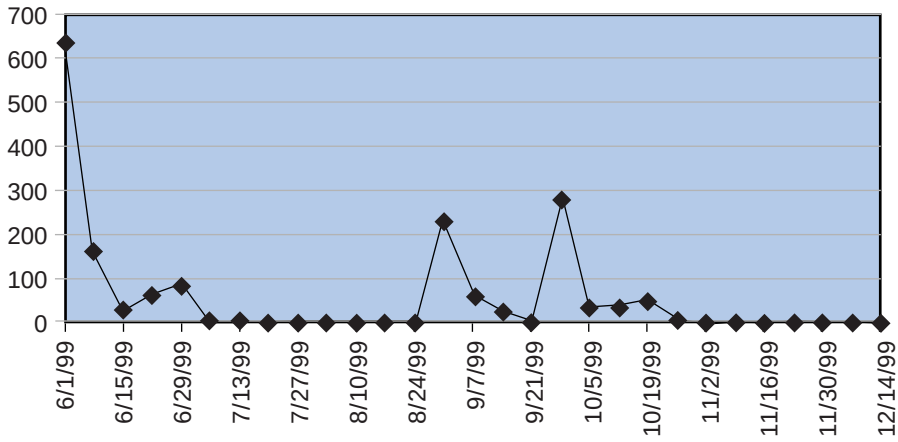
Sample Plots of Porting Indicators

This section includes example plots of porting indicator data described in this application note.

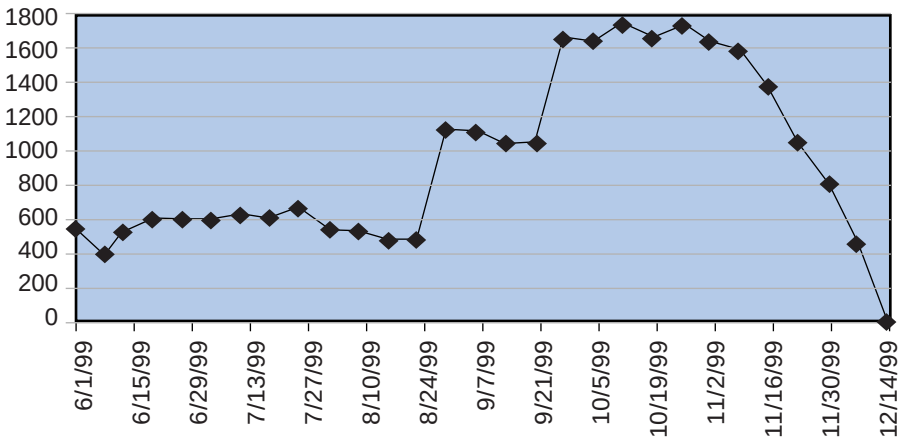
Total Number of Files Compiled (IA-64)

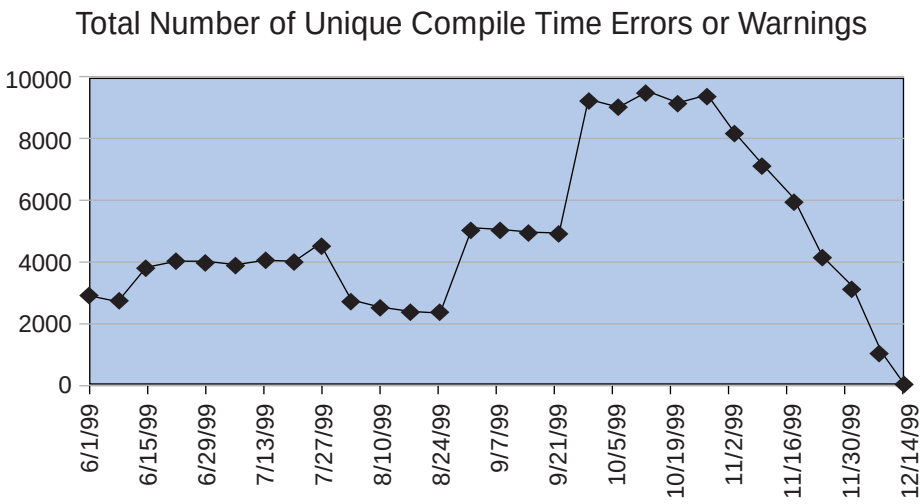


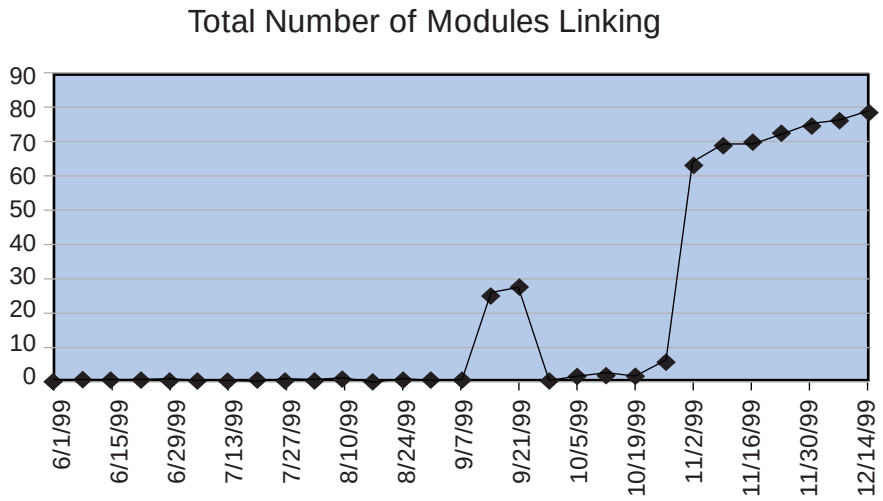
Total Number of Compile Time Errors (IA-64)



Total Number of Files with Compile Time Errors and Warnings




Total Number of Unique Compile Time Errors or Warnings
10000
8000
6000
4000
2000
0
6/1/99
6/15/99
6/29/99
7/13/99
7/27/99
8/10/99
8/24/99
9/7/99
9/21/99
10/5/99
10/19/99
11/2/99
11/16/99
11/30/99
12/14/99


Total Number of Modules Linking
90
80
70
60
50
40
30
20
10
0
6/1/99
6/15/99
6/29/99
7/13/99
7/27/99
8/10/99
8/24/99
9/7/99
9/21/99
10/5/99
10/19/99
11/2/99
11/16/99
11/30/99
12/14/99

Summary

This application note presents five different indicators that can be used to help track the progress of a porting project. These indicators are easy to collect, objective, and provide a useful way to track your porting effort. This application note also provides a Perl script to help collect the indicator data.